IMT Nord Europe
École Mines-Télécom
IMT-Université de Lille

Graph Convolutional Networks (GCNs)

Introduction to GCNs

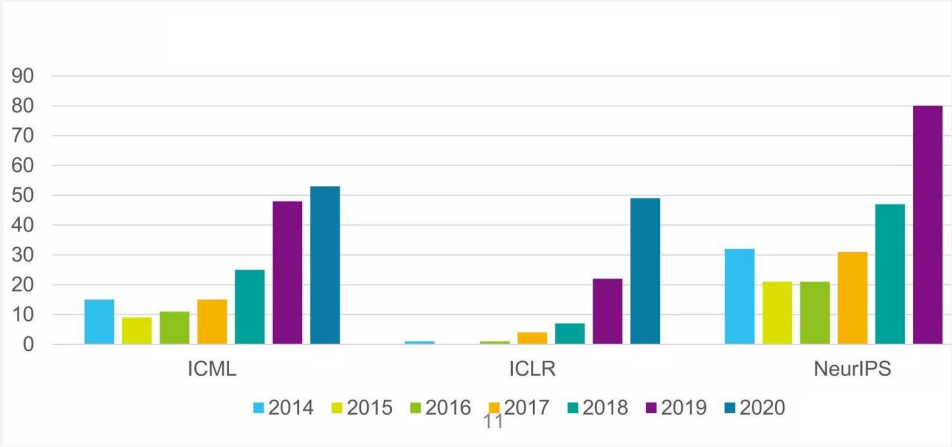
Ikne Omar

01.12.2023

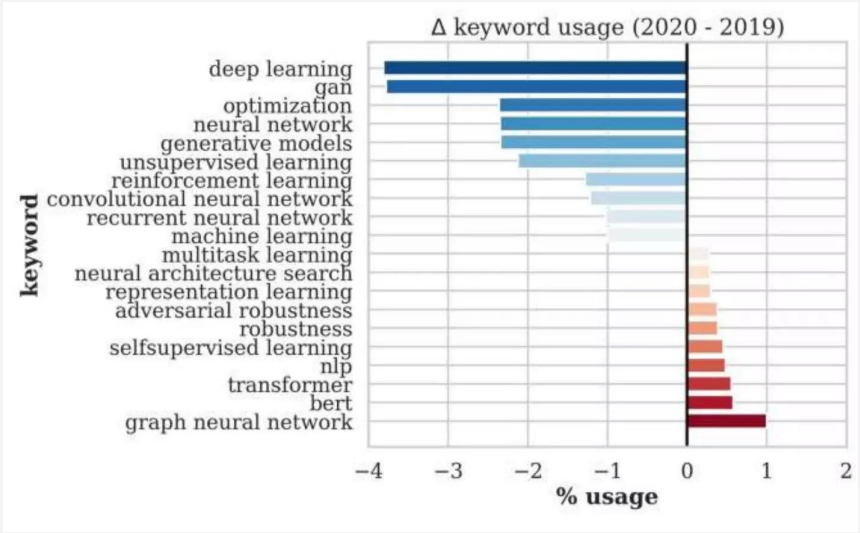
Contents

First, a few statistics

Number of accepted papers
Title: “Graph”



ICLR submissions

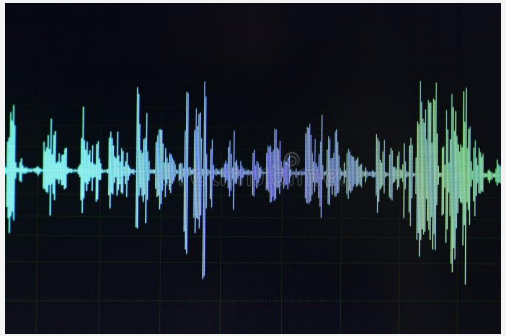


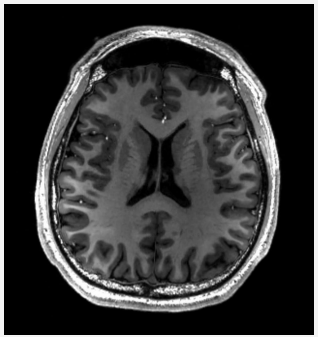
- ➔ *ICML:* International Conference on Machine Learning (rank A)
- ➔ *ICLR:* International Conference on Learning Representations (rank A*)
- ➔ *NeurIPS:* Neural Information Processing Systems (rank A*)

1. Why graphs ?

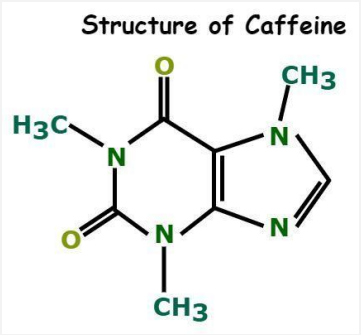
Understanding Connectivity: Explore data beyond tabular and image formats

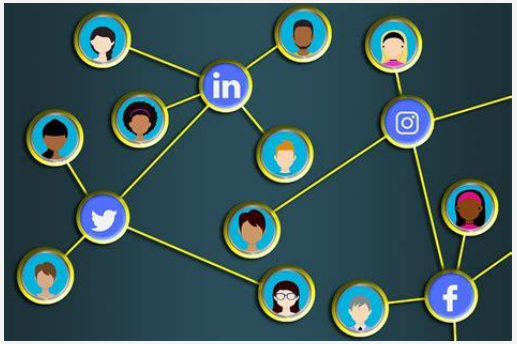
Image & sequential data…

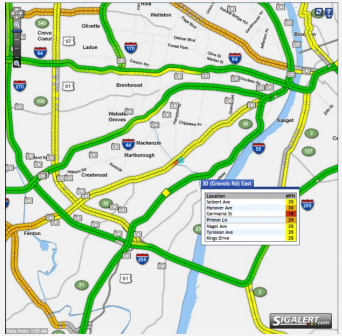




Graph structure data…



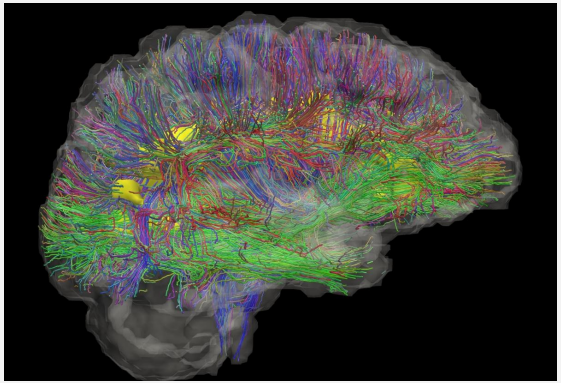


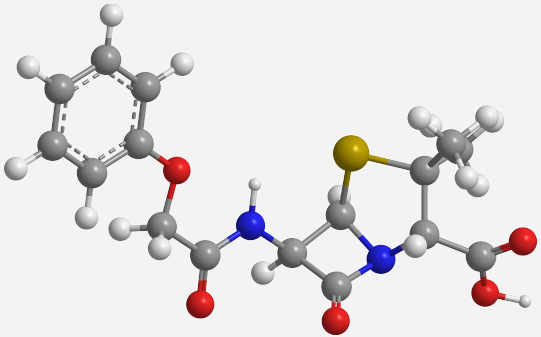


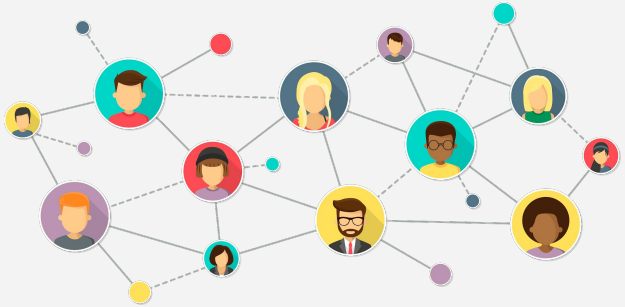
Graphs are everywhere !

Why Graphs?

Many problems are graphs in true nature. In our world, we see many data are graphs, such as molecules, social networks, and paper citations networks.

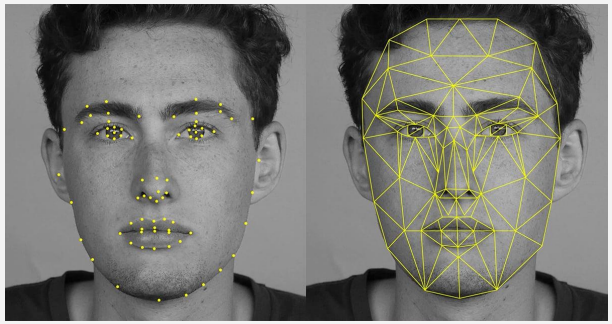


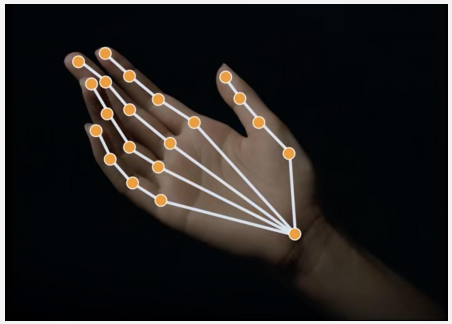




Graphs are everywhere !

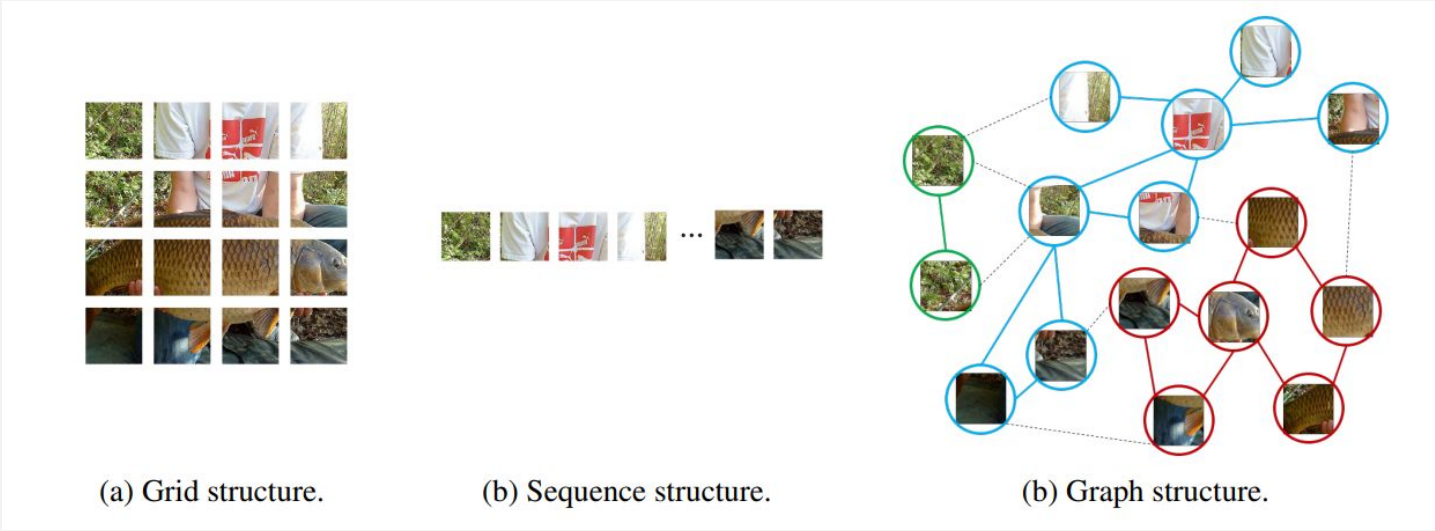
Even humans can be seen as graphs !





Graphs are everywhere !

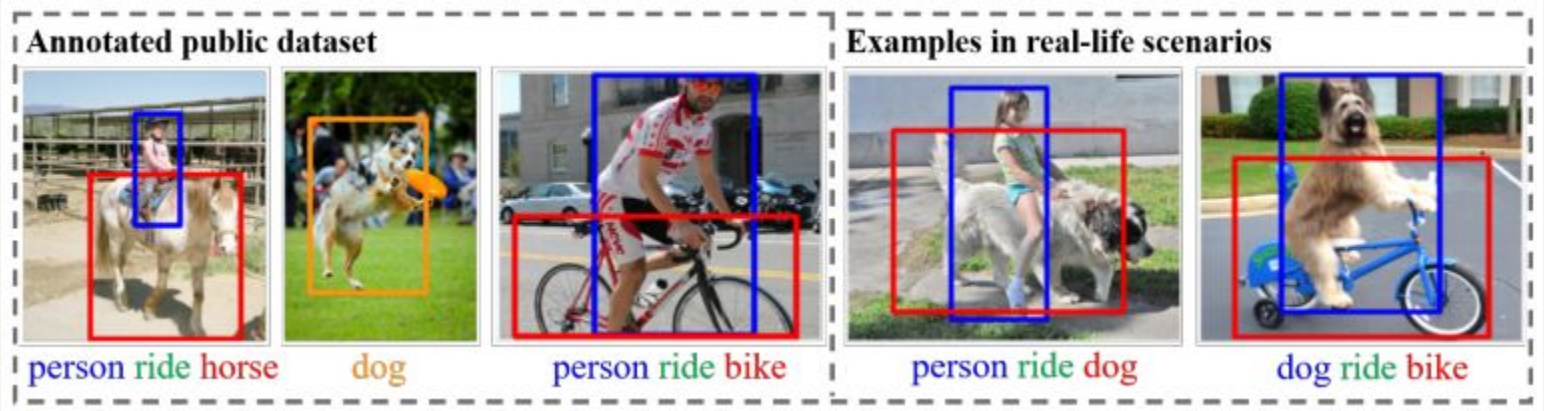
Even more surprising! Any image can be considered a graph!

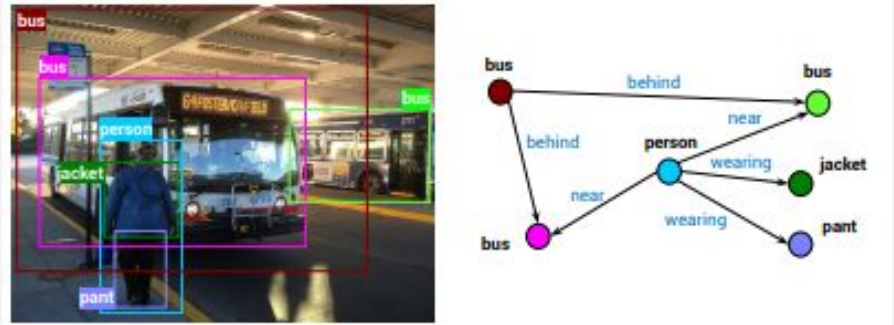


(a) Grid structure. (b) Sequence structure. (b) Graph structure.

Graphs are everywhere !

Graphs are very useful in scene understanding





In a scene graph of an image:

- Nodes: objects
- Edges: relation between objects

2. Fundamentals of Graph Structure

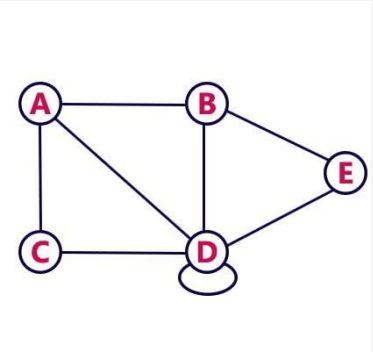
Graphs: Back to basics

What is a graph ?

A graph *G=(V, E)* is a set of points, called **nodes** or **vertices (V)**, which are interconnected by a set of lines called **edges (E)**.

$$V = \{v_1, v_2, v_3, \ldots, v_n\}$$
$$E = \{(v_i, vj), \ldots\}$$

- Directed / Undirected
- Weighted / Unweighted



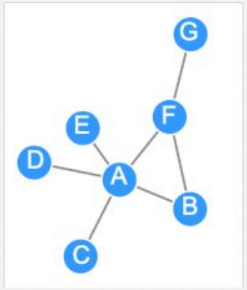
What is an adjacency matrix ?

The adjacency matrix, also called the connection matrix, is a matrix containing rows and columns which is used to represent a graph connections, with 0 or 1 in the position of (*vi* , *vj*) according to the condition whether *vi* and *vj* are adjacent or not.

	A	B	C	D	E
A	0	1	1	1	0
B	1	0	0	1	1
C	1	0	0	1	0
D	1	1	1	1	1
E	0	1	0	1	0

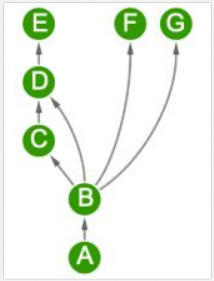
Graphs: Back to basics

Undirected (unweighted)



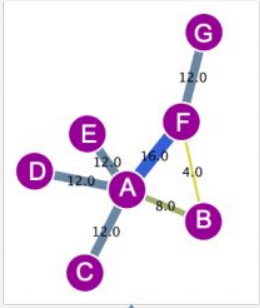
	A	B	C	D	E	F	G
A	0	1	1	1	1	1	0
B	1	0	0	0	0	1	0
C	1	0	0	0	0	0	0
D	1	0	0	0	0	0	0
E	1	0	0	0	0	0	0
F	1	1	0	0	0	0	1
G	0	0	0	0	0	1	0

Directed (unweighted)



	A	B	C	D	E	F	G
A	0	1	0	0	0	0	0
B	0	0	1	1	0	1	1
C	0	0	0	1	0	0	0
D	0	0	0	0	1	0	0
E	0	0	0	0	0	0	0
F	0	0	0	0	0	0	0
G	0	0	0	0	0	0	0

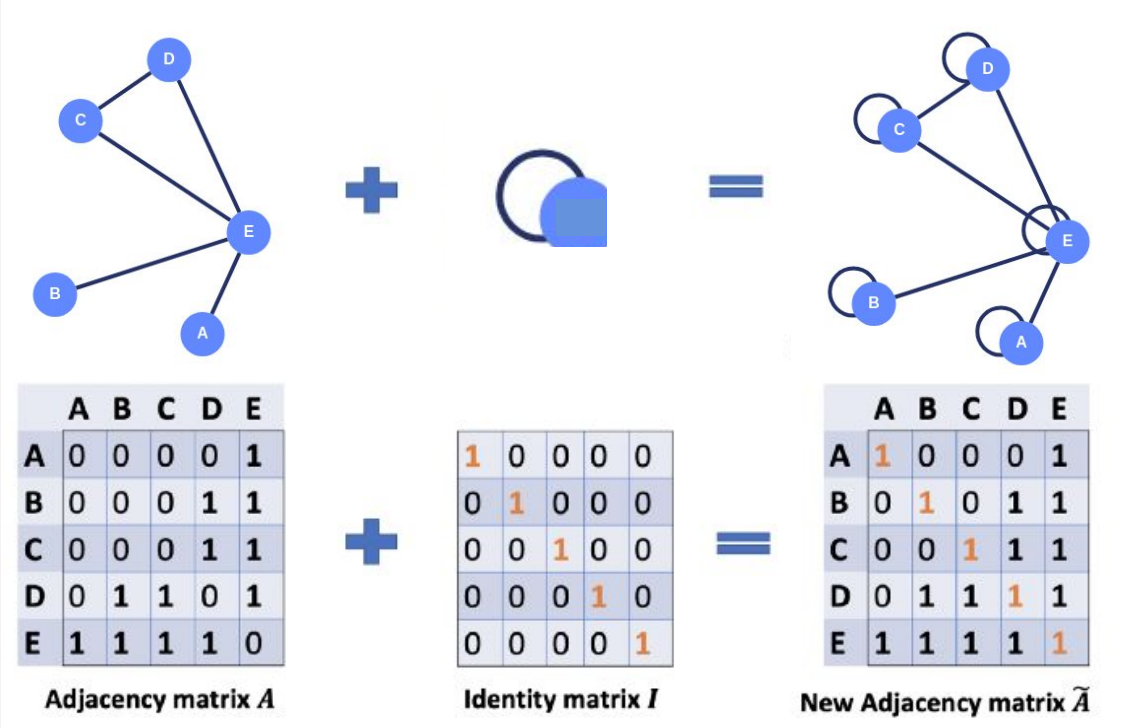
Weighted (undirected)



	A	B	C	D	E	F	G
A	0	8	12	12	12	16	12
B	8	0	0	0	0	4	0
C	12	0	0	0	0	0	0
D	12	0	0	0	0	0	0
E	12	0	0	0	0	0	0
F	16	4	0	0	0	0	12
G	12	0	0	0	0	12	0

Graphs: Back to basics

Self-loops: Incorporating Node Information



	A	B	C	D	E
A	0	0	0	0	1
B	0	0	0	1	1
C	0	0	0	1	1
D	0	1	1	0	1
E	1	1	1	1	0

Adjacency matrix A

1	0	0	0	0
0	1	0	0	0
0	0	1	0	0
0	0	0	1	0
0	0	0	0	1

Identity matrix I

	A	B	C	D	E
A	1	0	0	0	1
B	0	1	0	1	1
C	0	0	1	1	1
D	0	1	1	1	1
E	1	1	1	1	1

New Adjacency matrix $\tilde{A}$

Graphs: Graph Data

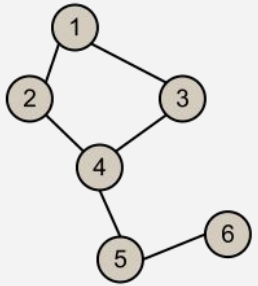
Edges: Adjacency Matrix (A)

	1	2	3	4	5	6
1	0	1	1	0	0	0
2	1	0	0	1	0	0
3	1	0	0	1	0	0
4	0	1	1	0	1	0
5	0	0	0	1	0	1
6	0	0	0	0	1	0

```
array([[0, 1, 1, 0, 0, 0],
       [1, 0, 0, 1, 0, 0],
       [1, 0, 0, 1, 0, 0],
       [0, 1, 1, 0, 1, 0],
       [0, 0, 0, 1, 0, 1],
       [0, 0, 0, 0, 1, 0]])
```

A: shape(6, 6)

Nodes: Node Feature Matrix (X)




```
array([[ 0.5 , -0.14,  0.65,  1.52],   1
       [-0.23, -0.23,  1.58,  0.77],   2
       [-0.47,  0.54, -0.46, -0.47],   3
       [ 0.24, -1.91, -1.72, -0.56],   4
       [-1.01,  0.31, -0.91, -1.41],   5
       [ 1.47, -0.23,  0.07, -1.42]])  6
```

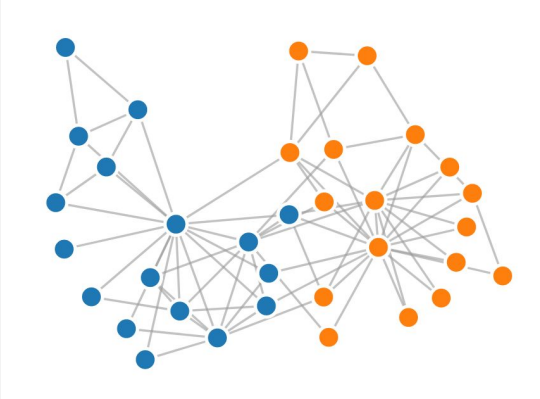
X: shape(6, 4)

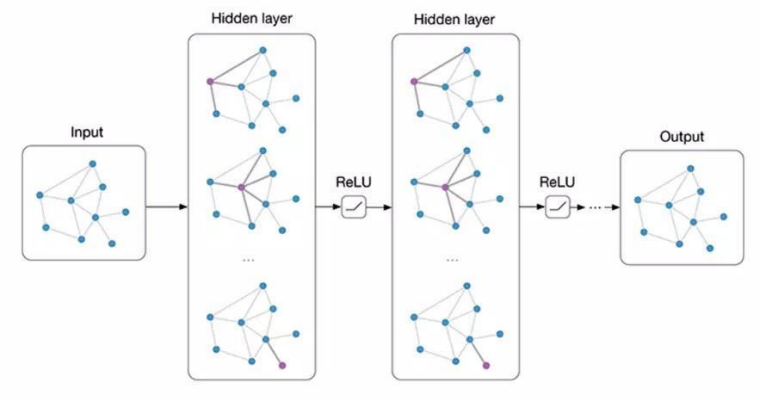
Graphs: Frameworks & utils

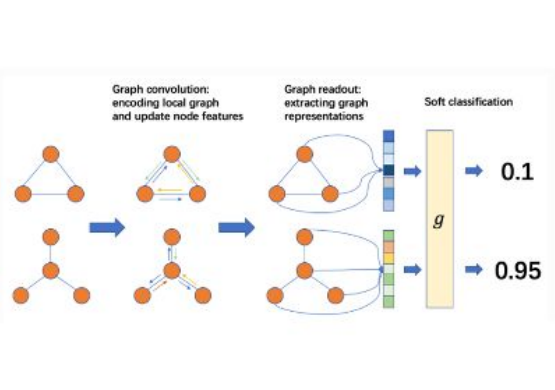
NetworkX
Network Analysis in Python

PyTorch
geometric

DEEP
GRAPH
LIBRARY







- Store and mutate graphs
- Predefined graph algorithms:
  - Shortest path
  - Dijkstra
  - Clustering
  - Centrality
- Network analysis
- Visualization tools

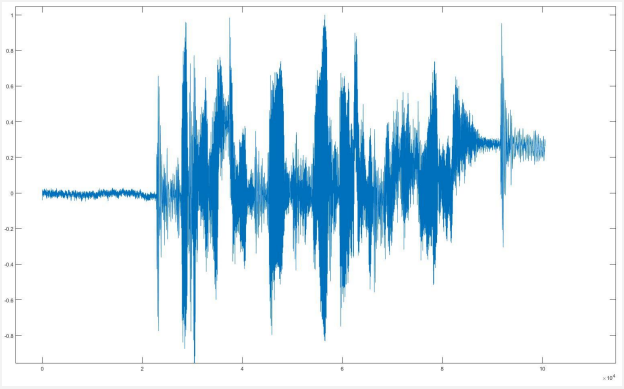
- An extension library for pyTorch
- Top-rated papers use it for implementation
- Long list of ready-to-use methods and algos:
  - *TransformerConf (2020)*
  - *GCN2Conv (2020)*
  - *DeeperGCN (2020)*
  - *…*

- Building Deep Learning models
- Great tutorials
- Generative graphs
- Very useful for research and advanced tasks
- Supports pyTorch and TensorFlow

3. From CNNs to GCNs

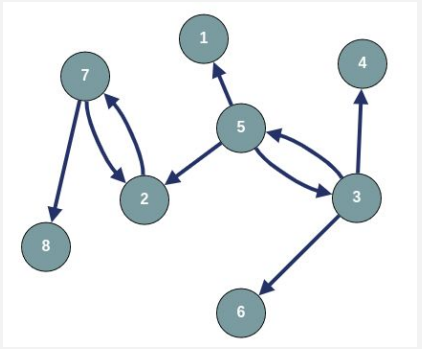
Image/Sequence vs. Graph Data

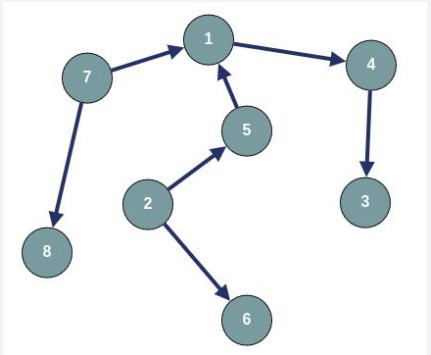
Sequential & image data…

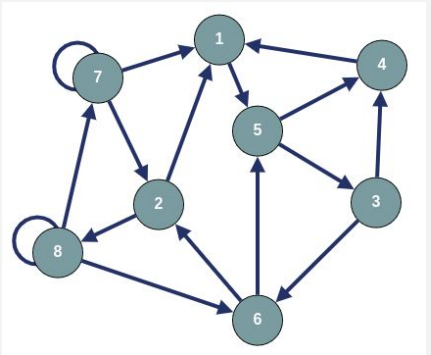


Graphs	are	everywhere	!

Graph-like data…







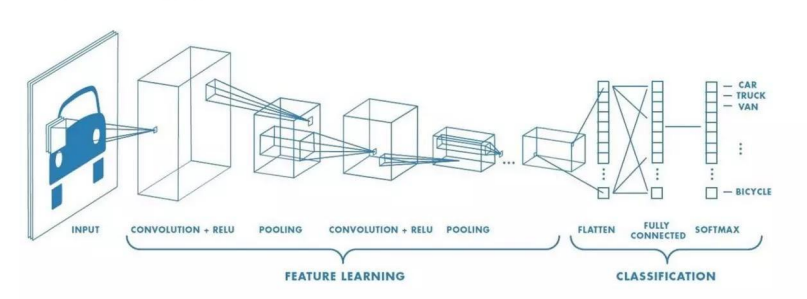
From CNNs to GCNs

- CNNs leverage the convolutional operator to extract the spatial regularity of the images.

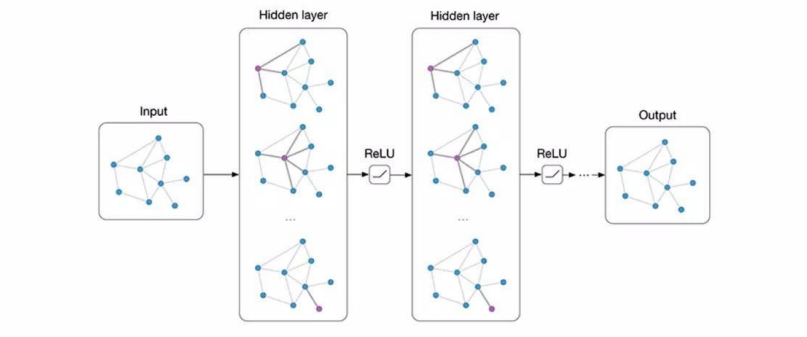
- Graphs can be seen as a strict generalisation of images.
  - Consider any image as a grid graph.
  - Each pixel represents a node
  - Each pixel is adjacent to its 4/8 neighbours

- *Can we adapt the convolutions to operate on graph data ?*

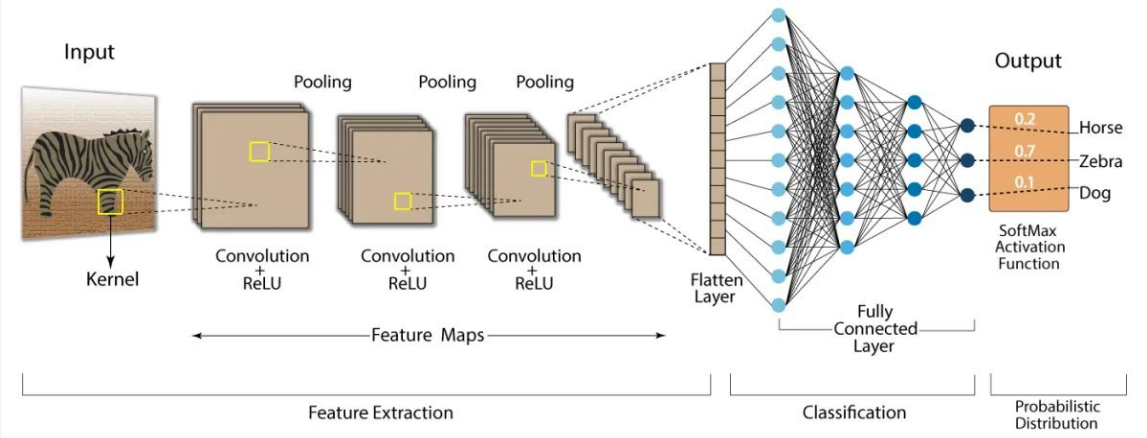
CNNs



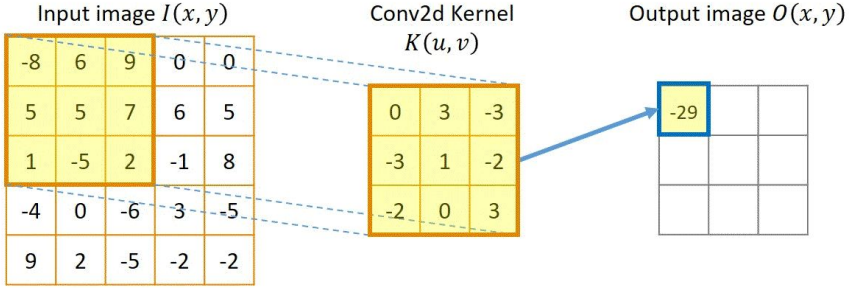
GCNs



CNN: Convolution on Images



Convolution Neural Network (CNN)



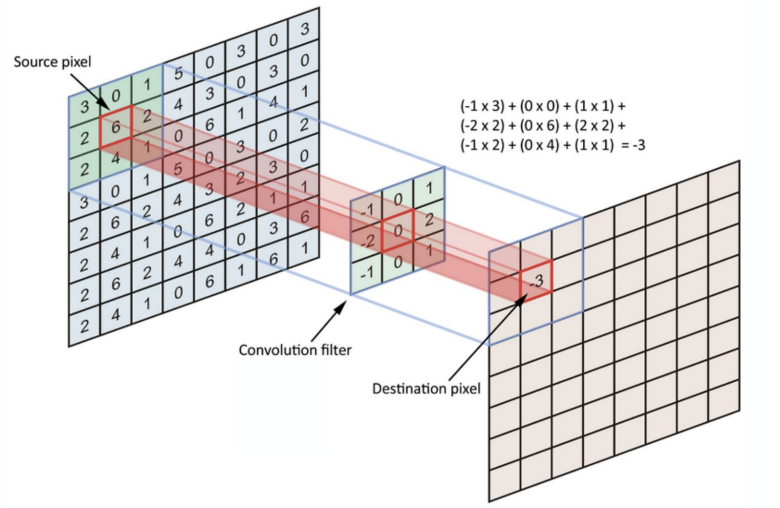
Update Rule:

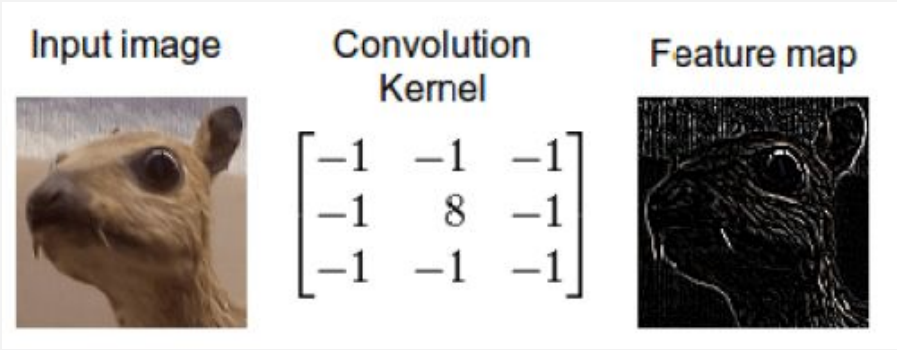
$$H^{(l+1)} = \sigma\left(\sum_{i,j\in\{-1,0,1\}} H^{(l)}_{x+i,y+j} \times W^{(l+1)}_{i,j}\right)$$

shift input kernel

CCN: Convolution on Images

Convolutions capture local patterns and features in images.





CCN: Convolution on Images

Convolutions Parameters:

Parameters

- **in_channels** (*int*) – Number of channels in the input image
- **out_channels** (*int*) – Number of channels produced by the convolution
- **kernel_size** (*int or tuple*) – Size of the convolving kernel
- **stride** (*int or tuple, optional*) – Stride of the convolution. Default: 1
- **padding** (*int or tuple, optional*) – Zero-padding added to both sides of the input. Default: 0
- **padding_mode** (*string, optional*) – `'zeros'`, `'reflect'`, `'replicate'` or `'circular'`. Default: `'zeros'`
- **dilation** (*int or tuple, optional*) – Spacing between kernel elements. Default: 1
- **groups** (*int, optional*) – Number of blocked connections from input channels to output channels. Default: 1
- **bias** (*bool, optional*) – If `True`, adds a learnable bias to the output. Default: `True`

Shape:

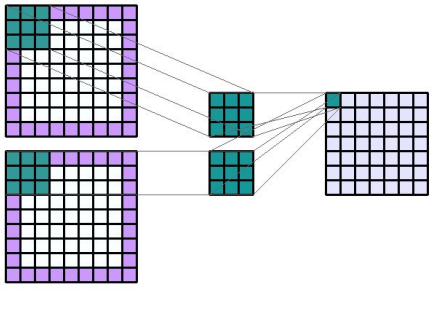
- Input: $(N, C_{in}, H_{in}, W_{in})$
- Output: $(N, C_{out}, H_{out}, W_{out})$ where

$$H_{out} = \left\lfloor \frac{H_{in} + 2 \times \text{padding}[0] - \text{dilation}[0] \times (\text{kernel\_size}[0] - 1) - 1}{\text{stride}[0]} + 1 \right\rfloor$$

$$W_{out} = \left\lfloor \frac{W_{in} + 2 \times \text{padding}[1] - \text{dilation}[1] \times (\text{kernel\_size}[1] - 1) - 1}{\text{stride}[1]} + 1 \right\rfloor$$

Stride *(stride = (1, 3))*

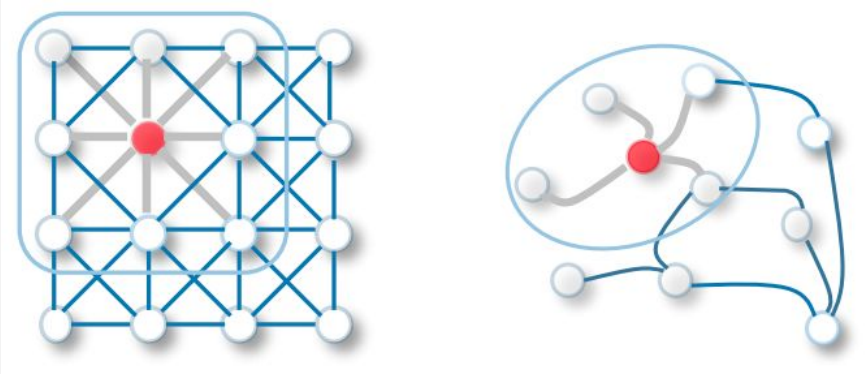
Padding *(padding = (1, 1))*



Input: (2, 7, 7) — Output: (1, 7, 7) — K : (3, 3) — P : (1, 1) — S : (1, 1)

Input: (3, 9, 9) — Output : (2, 7, 3) — K : (3, 3) — S : (1, 3)

Convolution From Image to Graph ?



Convolution on images vs. on graphs

On images

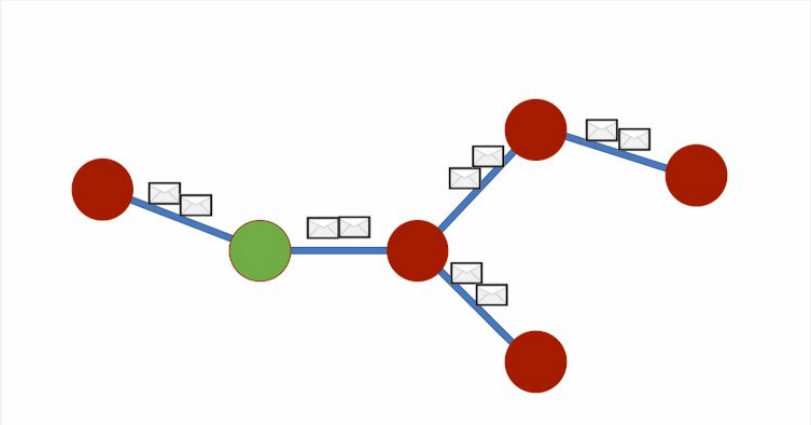
- Convolution works well on Euclidean data, such as image data.
- Convolutions capture local patterns and features in images.
- Convolutions provide a certain level of translation invariance

What about graphs ?

- Graphs are non-Euclidean structured data! i.e. they have no “axes”.
- What is a coordinate? And what is a shift?
- Convolutions seem hard to apply to graphs!

GCN: Convolution on graphs

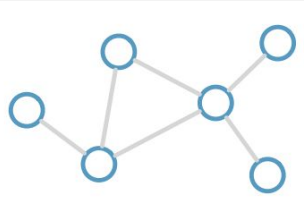
- Convolution can be referred to as Message Passing in GCNs.
- Message passing is performed in the context of a node's local neighborhood.
- Each node shares its current state with its neighbors, and each node’s new features are updated accordingly.



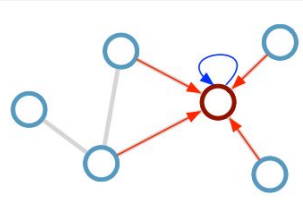
Message Passing

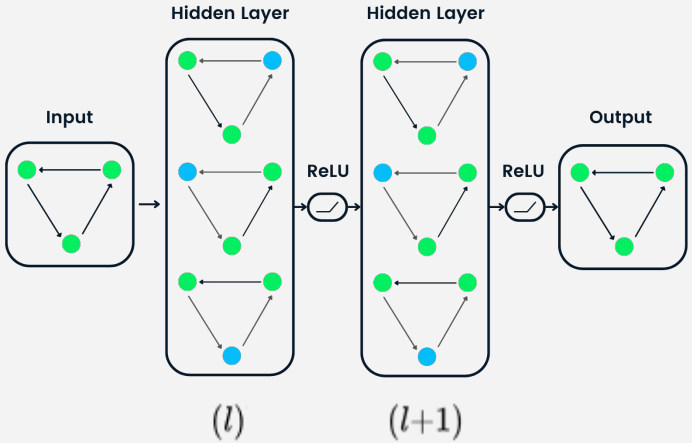
GCN: Convolution on graphs

Consider this undirected graph:



Calculate update for node in red:





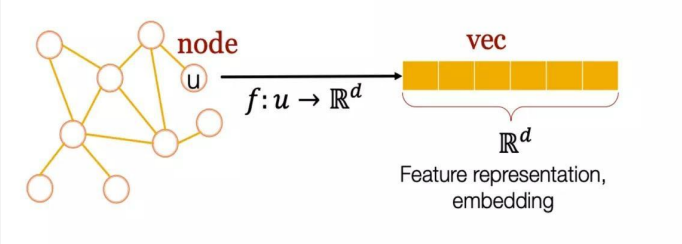
Update Rule:

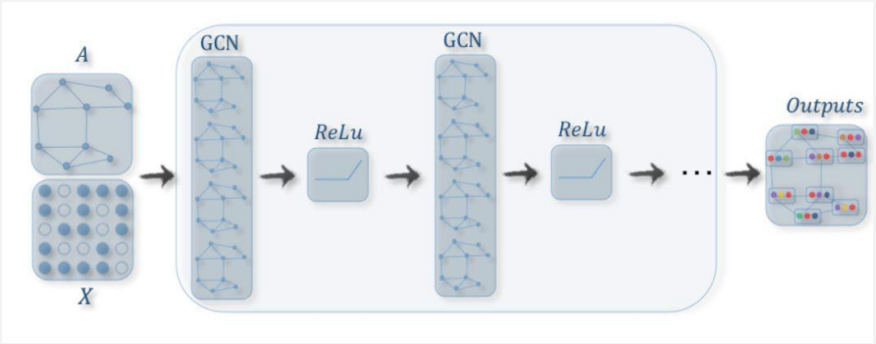
$$H^{(l+1)} = \sigma\left(H_0^{(l)}W^{(l)} + \sum_{j\in\mathcal{N}_i} H_j^{(l)}W^{(l)} + b^{(l)}\right) \quad \textbf{OR} \quad H^{(l+1)} = \sigma\left(AH^{(l)}W^{(l)} + b^{(l)}\right)$$

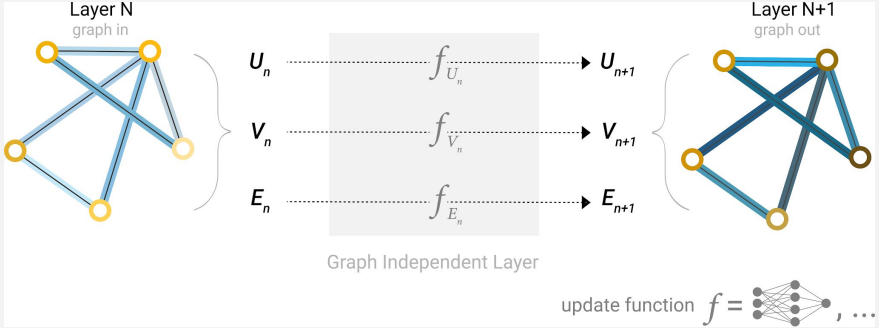
GCN: Convolution on graphs

Why learning embeddings/representations ?

- Projection to a Higher-Dimensional Space
- The higher-dimensional space provides a richer representation that facilitates the learning of complex relationships and patterns in the graph-structured data.
- The final layer's output can be seen as an embedding of nodes in a latent space, where nodes with similar roles or functions in the graph are close together.







GCN: Convolution on graphs

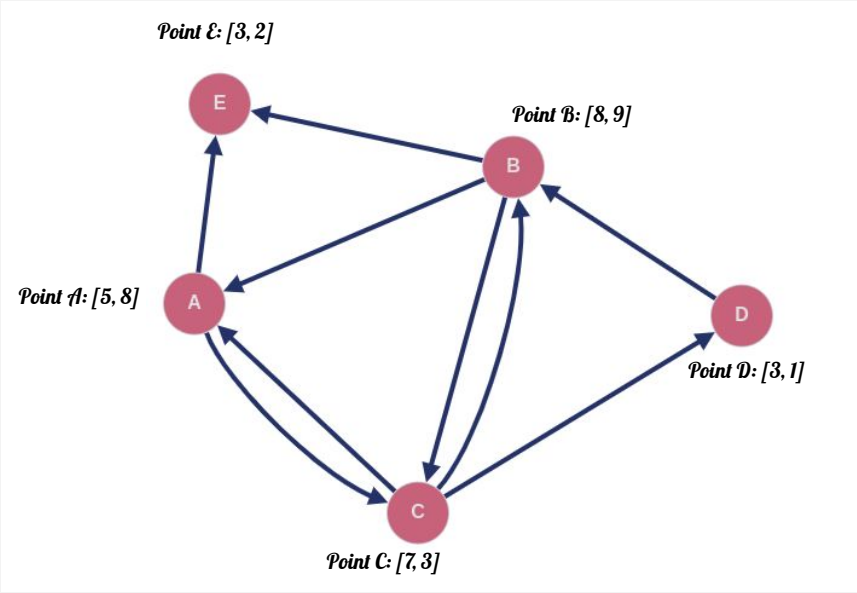
Let's take a look at an example!

Do the maths !

Given the following paper citation graph. Write down the corresponding adjacency matrix A and the features matrix X.

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}; \; X = \begin{bmatrix} 5 & 8 \\ 8 & 9 \\ 7 & 3 \\ 3 & 1 \\ 3 & 2 \end{bmatrix}; \; W = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

$$AXW = (AX)W$$



Simplified graph of a traffic network

Point X: [road capacity, traffic jam factor]

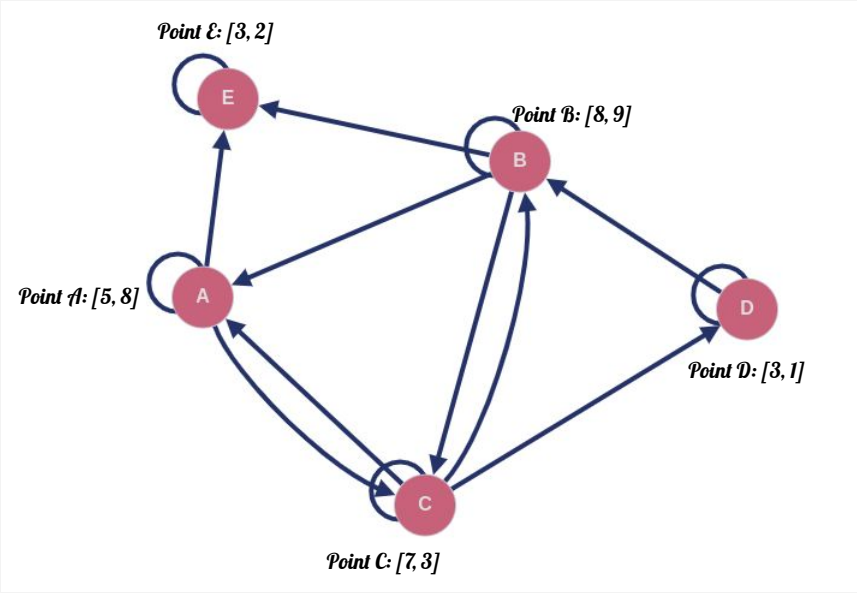
$$AX = \begin{bmatrix} 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} 5 & 8 \\ 8 & 9 \\ 7 & 3 \\ 3 & 1 \\ 3 & 2 \end{bmatrix} = \begin{bmatrix} 10 & 5 \\ 15 & 13 \\ 16 & 18 \\ 8 & 9 \\ 0 & 0 \end{bmatrix} \Longrightarrow$$

$$AXW = (AX)W = \begin{bmatrix} 10 & 5 \\ 15 & 13 \\ 16 & 18 \\ 8 & 9 \\ 0 & 0 \end{bmatrix} \times \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 15 & 15 & 15 & 15 \\ 28 & 28 & 28 & 28 \\ 34 & 34 & 34 & 34 \\ 17 & 17 & 17 & 17 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

GCN: Convolution on graphs

If we add self-loops !

$$A = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}; \; X = \begin{bmatrix} 5 & 8 \\ 8 & 9 \\ 7 & 3 \\ 3 & 1 \\ 3 & 2 \end{bmatrix}; \; W = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$



Simplified graph of a traffic network

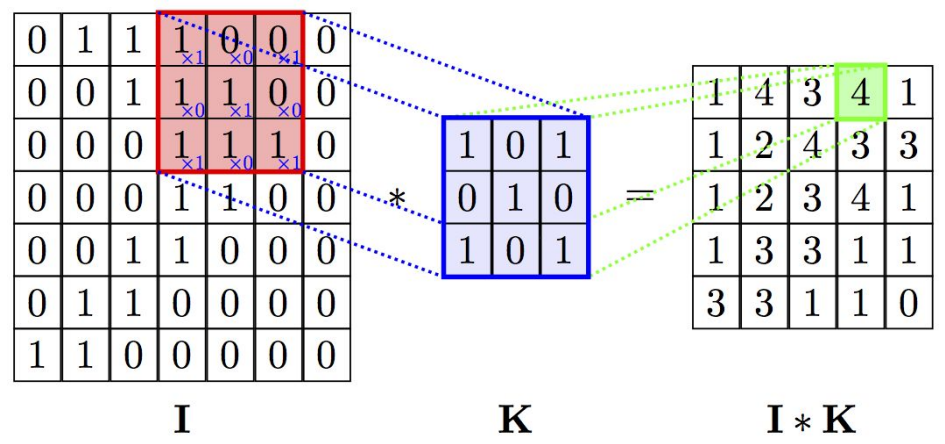
$$AXW = (AX)W$$

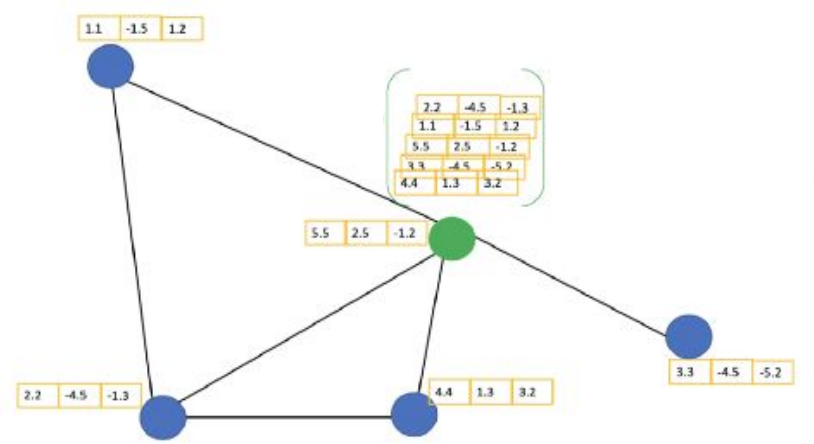
Point X: [road capacity, traffic jam factor]

$$AX = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 5 & 8 \\ 8 & 9 \\ 7 & 3 \\ 3 & 1 \\ 3 & 2 \end{bmatrix} = \begin{bmatrix} 15 & 13 \\ 23 & 22 \\ 23 & 21 \\ 11 & 10 \\ 3 & 2 \end{bmatrix} \Longrightarrow AXW = (AX)W = \begin{bmatrix} 15 & 13 \\ 23 & 22 \\ 23 & 21 \\ 11 & 10 \\ 3 & 2 \end{bmatrix} \times \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 28 & 28 & 28 & 28 \\ 45 & 45 & 45 & 45 \\ 44 & 44 & 44 & 44 \\ 21 & 21 & 21 & 21 \\ 5 & 5 & 5 & 5 \end{bmatrix}$$

To sum up, convolutional operator for Image vs. Graph

Convolutions on images vs. on graphs



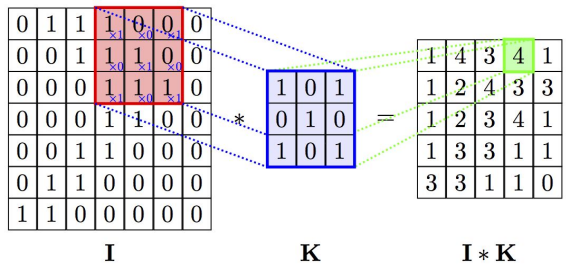


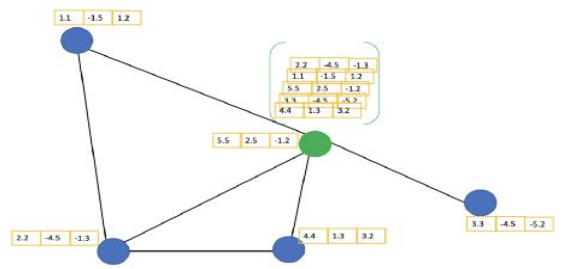
$$H^{(l+1)} = \sigma\left(\sum_{i,j\in\{-1,0,1\}} H^{(l)}_{x+i,y+j} \times W^{(l+1)}_{i,j}\right)$$

$$H^{(l+1)} = \sigma\left(AH^{(l)}W^{(l)} + b^{(l)}\right)$$

To sum up, convolutional operator for Image vs. Graph

Convolutions on images vs. on graphs





Local Receptive Fields:

- Convolutional operations are applied using small filters (kernels) that slide over the input image.
- Each filter looks at a local receptive field (a small region of the image).

Parameter Sharing:

- Parameters (weights) of the filters are shared across the entire image.
- The same set of weights is applied to different local receptive fields.

Aggregation of Information:

- Aggregate information by computing a weighted sum of the values in the receptive field.

Local Neighborhoods:

- Message passing occurs over the graph structure where nodes represent entities, and edges represent relationships.
- Message passing is performed in the context of a node's local neighborhood.

Parameter Sharing:

- GCNs use learnable parameters (weights) associated with edges to determine how information is shared between connected nodes.
- The same set of parameters is shared across all nodes in the graph.

Aggregation of Information:

- Aggregation is typically performed by computing a weighted sum of neighbor node features.

GCNs: Convolutional Operator

Graph convolution implementation in *pytorch*:

```python
import torch.nn as nn

class GraphConvolution(nn.Module):
    """
    Simple GCN layer, similar to https://arxiv.org/abs/1609.02907
    """
    def __init__(self, in_features, out_features, bias=True):
        super(GraphConvolution, self).__init__()
        self.in_features = in_features
        self.out_features = out_features
        self.weight = Parameter(torch.FloatTensor(in_features, out_features))
        if bias:
            self.bias = Parameter(torch.FloatTensor(out_features))
        else:
            self.bias = None
        self.activation = nn.ReLU()

    def forward(self, input, adj):
        support = torch.mm(input, self.weight)
        output = torch.spmm(adj, support)
        if self.bias is not None:
            output = output + self.bias
        return self.activation(output)
```

$$H^{(l+1)} = \sigma\left(AH^{(l)}W^{(l)} + b^{(l)}\right)$$

$H^{(l)}W^{(l)}$

$AH^{(l)}W^{(l)}$

$AH^{(l)}W^{(l)} + b^{(l)}$

$\sigma\left(AH^{(l)}W^{(l)} + b^{(l)}\right)$

Shapes:

$$A \in \mathbb{R}^{n \times n}$$

$$H^{(l)} \in \mathbb{R}^{n \times d}$$

$$W^{(l)} \in \mathbb{R}^{d \times d'}$$

$$b^{(l)} \in \mathbb{R}^{d'}$$

- *n:* number of nodes
- *d:* in_features
- *d':* out_features

$$\Longrightarrow$$

$$AH^{(l)} \in \mathbb{R}^{n \times d}$$

$$AH^{(l)}W^{(l)} \in \mathbb{R}^{n \times d'}$$

$$AH^{(l)}W^{(l)} + b(l) \in \mathbb{R}^{n \times d'}$$

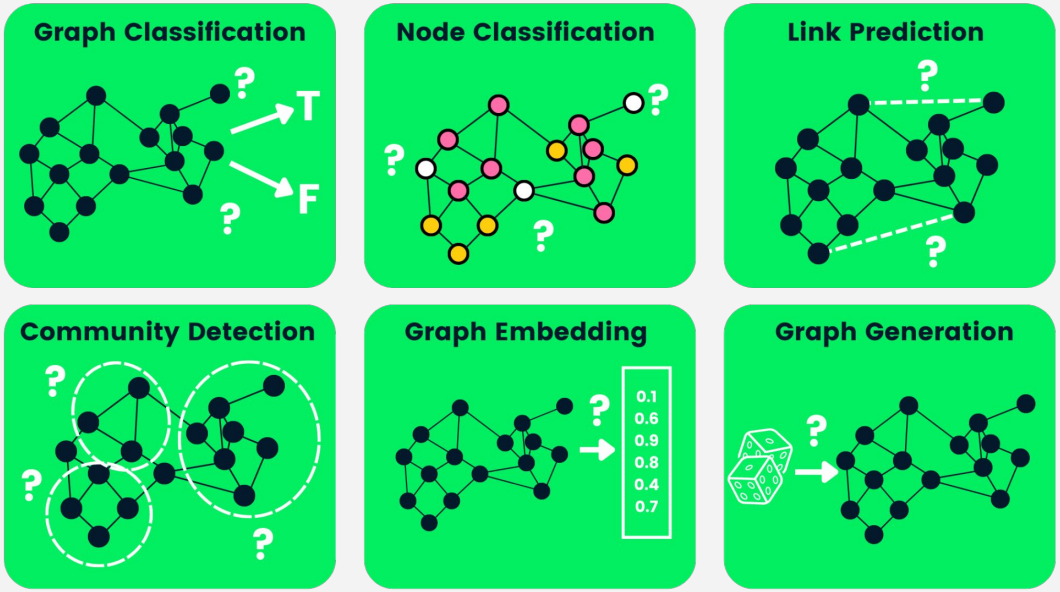
Mathematically:

$$f : \mathbb{R}^{d} \longrightarrow \mathbb{R}^{d'}$$

$$X \mapsto AXW + b$$

4. GCNs: Practical Aspects

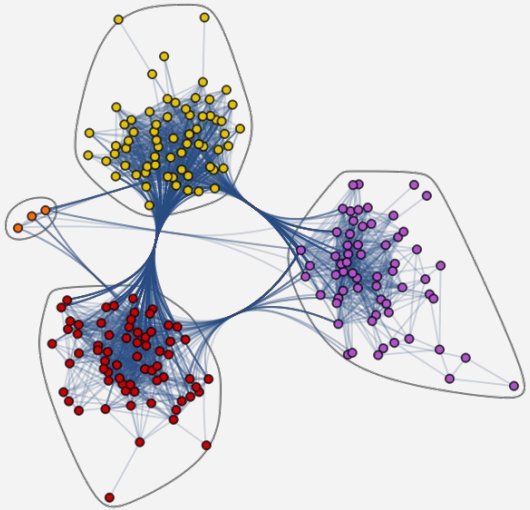
Tasks on Graphs



Tasks on Graphs: Node classification

➔ **Node classification:** Predict a type of a given node

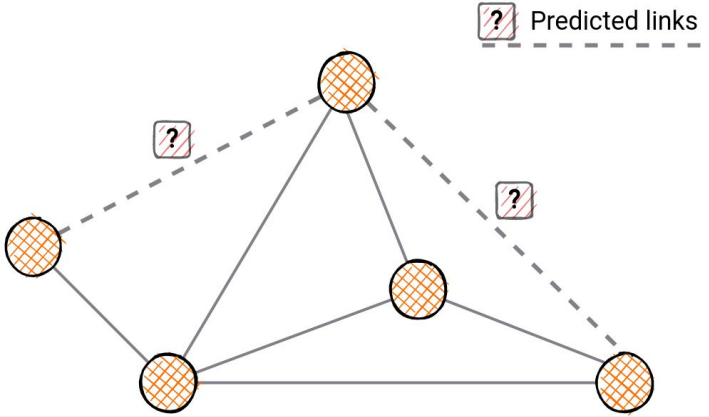
e.g., Malicious users in a social network, community clustering, etc



Tasks on Graphs: Link prediction

➔ **Link prediction:** Predict whether two nodes are linked

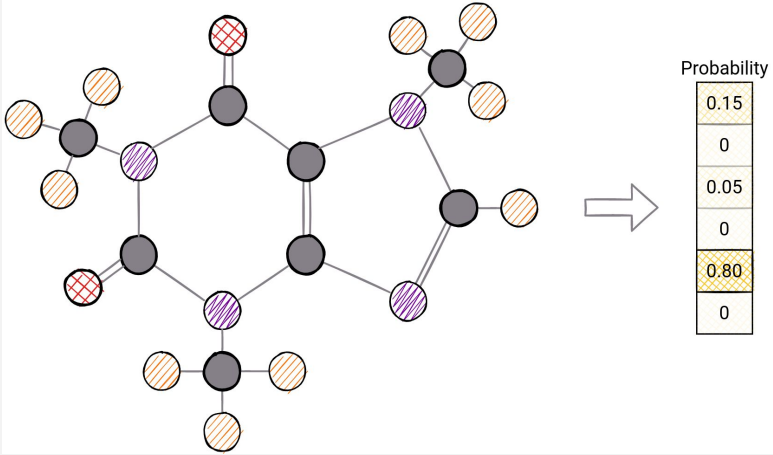
e.g., protein-protein interaction, friend suggestion, etc

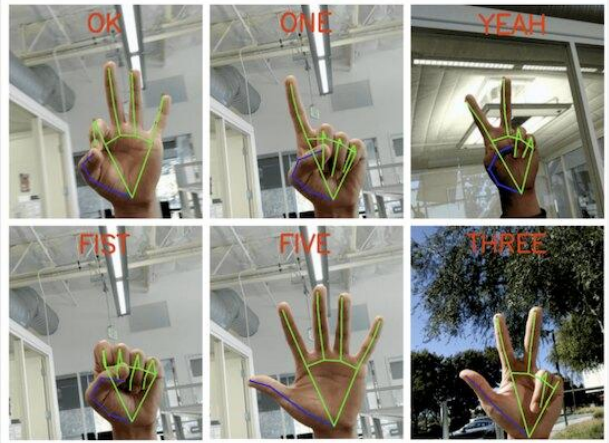


Tasks on Graphs: Graph classification

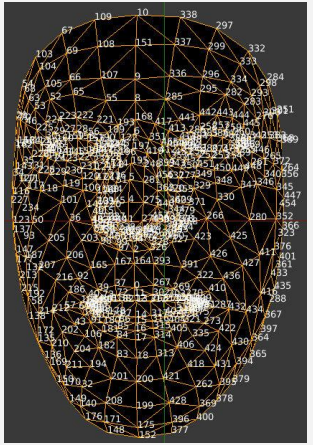
➔ **Graph classification:** Predict the class of a given graph

e.g., type of a chemical molecule, gesture recognition, etc.



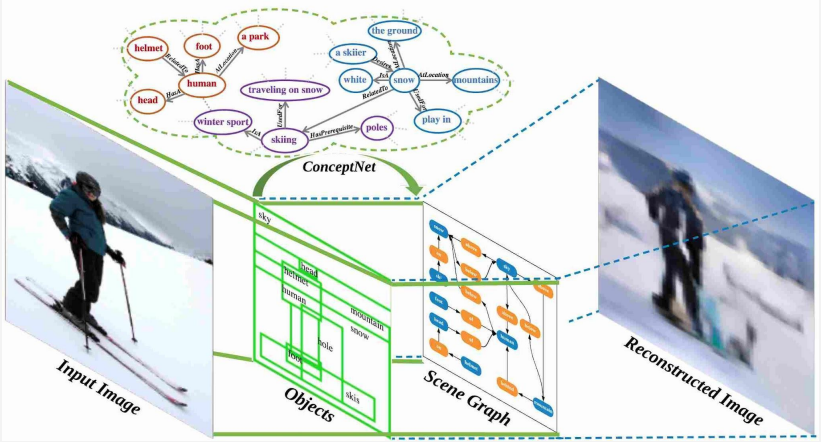


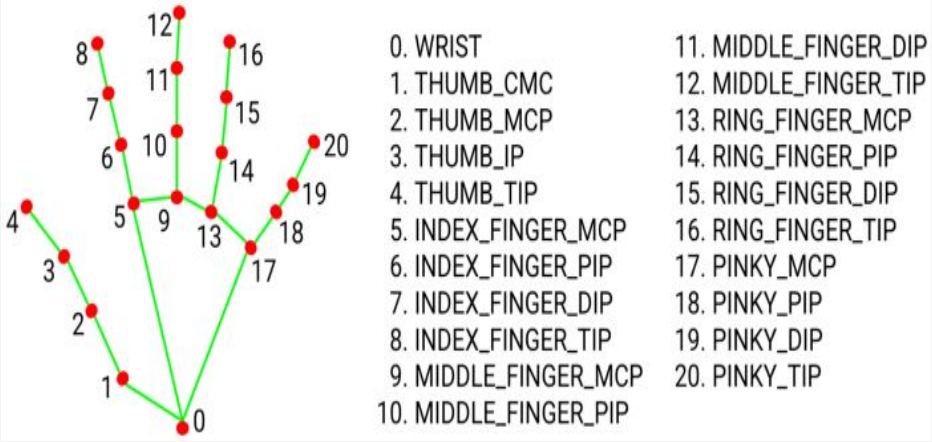
Tasks on Graphs: Graph Generation



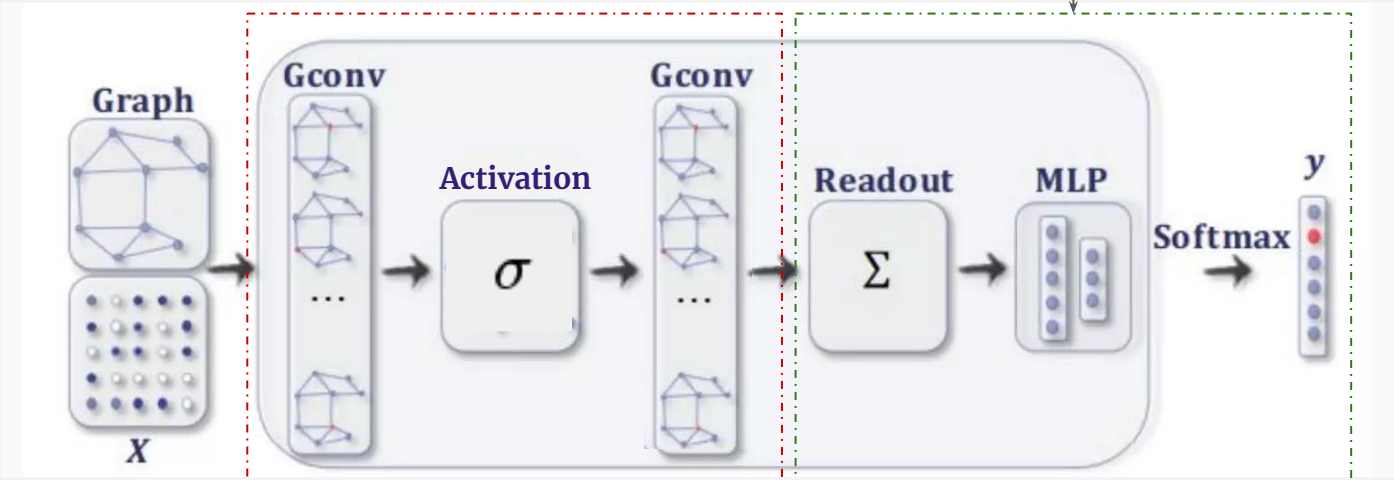
➔ **Graph generation:** generate a graph given an input data.

e.g., interactions learning, molecule generation, interactions sparsifying, etc.





Overall Structure of GCN

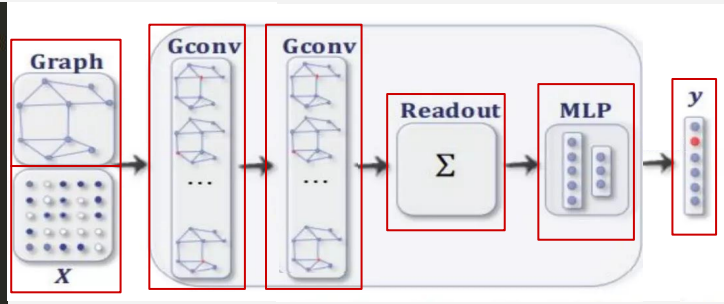


Depending on the downstream task, we adjust this part:

- Node classification
- Graph classification
- Graph generation
- etc.

Basic GCN Implementation

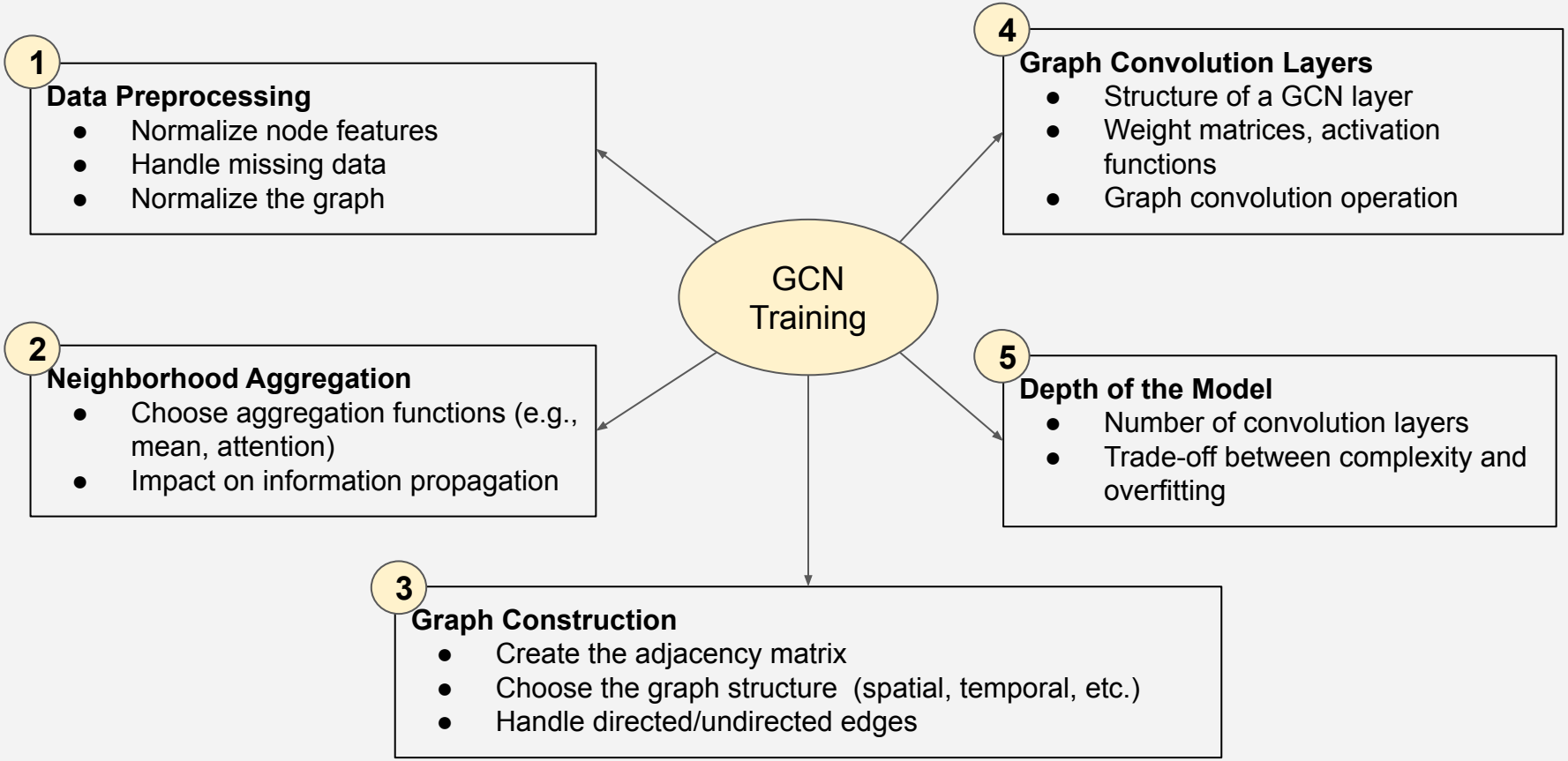
```python
class GCN(nn.Module):
    def __init__(self, in_dim, n_nodes, hidden_dim, n_gcn_layers, n_classes, dropout):
        super(GCN, self).__init__()
        ## GCN layers to learn graph representation
        self.gc1 = GraphConvolution(in_dim, hidden_dim)
        self.gc2 = GraphConvolution(hidden_dim, hidden_dim)

        ## MLP head to classify the graphs
        self.mlp = nn.Sequential(
            nn.Linear(hidden_dim * n_nodes, 64),
            nn.ReLU(),
            nn.Linear(64, n_classes)
        )

        ## dropout parameter to prevent overfitting
        self.dropout = dropout

    def forward(self, x, adj):

        ## first gcn layer with activation function and dropout
        x = F.relu(self.gc1(x, adj))
        x = F.dropout(x, self.dropout, training=self.training)

        ## second gcn layer with activation function and dropout
        x = F.relu(self.gc2(x, adj))
        x = F.dropout(x, self.dropout, training=self.training)

        ## reshape x from (b, n_joints, hidden_dim) into (b, n_joints * hidden_dim)
        b, _, _ = x.shape
        x = x.view(b, -1)

        ## mlp for classification
        out = self.mlp(x)

        return out
```



Parameters:

- *in_dim:* number of features of X
- *n_nodes:* number of nodes in the graph.
- *hidden_dim:* embedding dimension
- *n_gcn_layers:* number of convolutional layers
- *n_classes:* number of classes

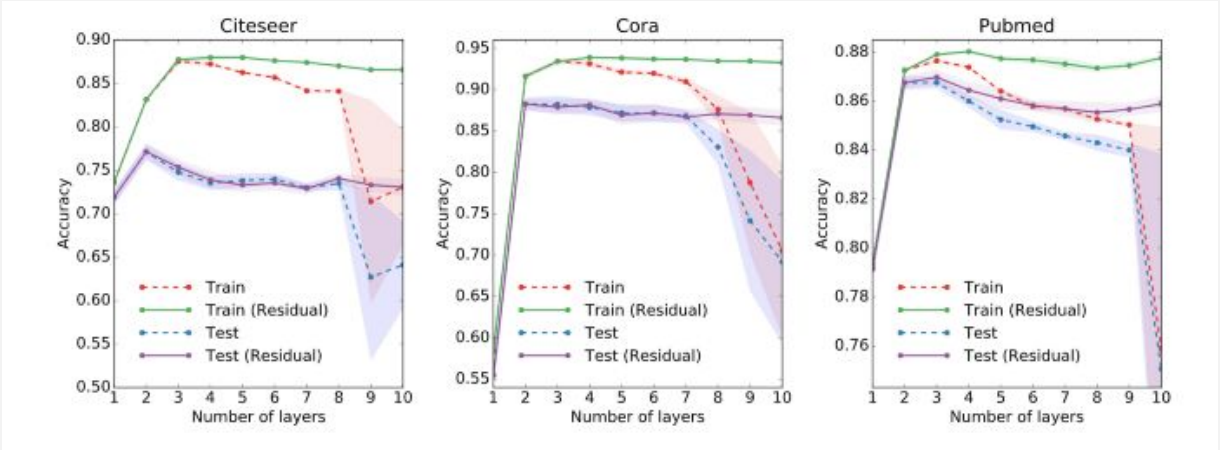
Training GCNs: Key considerations



Practical advices

Potential Pitfalls when going deep:

- Vanishing gradient
- Overfitting
- Over-smoothing: node vectors become too similar
- Bottleneck(over-squashing): a single node vector contains data of too many nodes



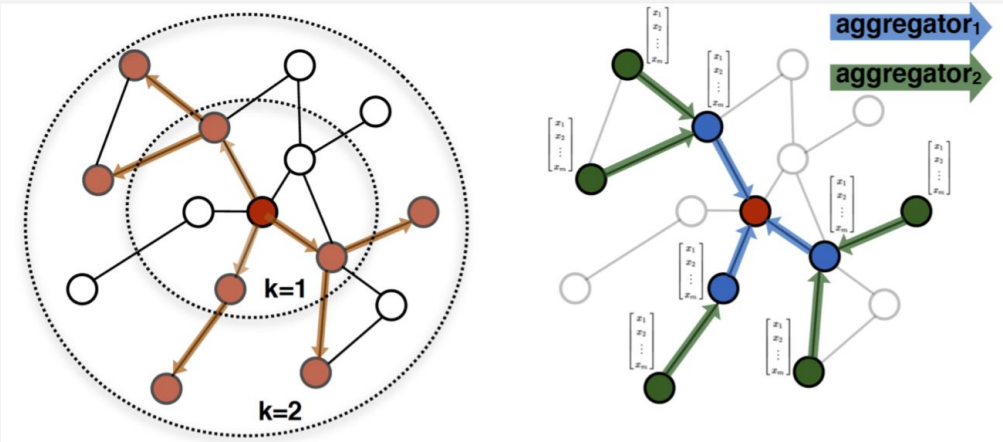
Model	2-Layer	4-Layer	8-Layer	16-Layer	32-Layer	64-Layer
GCN-res	88.18±1.59	86.50±1.87	84.83±1.93	78.60±4.28	59.82±7.74	39.71±5.15
PairNorm	79.98±3.80	82.32±2.79	81.52±3.66	82.29±2.62	81.91±2.45	81.72±2.82
NodeNorm	**89.53**±1.29	88.60±1.36	88.02±1.67	88.41±1.25	88.30±1.30	87.40±2.06

https://arxiv.org/abs/2006.07107 - Effective Training Strategies for Deep Graph Neural Networks

More advanced GCNs variantes

➔ **GraphSAGE (Graph Sample and Aggregated)**
"Inductive Representation Learning on Large Graphs" by Hamilton et al. (2017).

Key Idea:
GraphSAGE addresses scalability issues by sampling a fixed-size neighborhood around each node during training. It aggregates information from the sampled neighbors, allowing it to scale to large graphs.



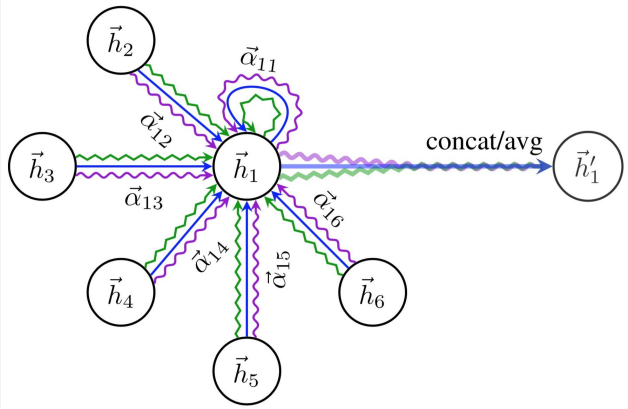
More advanced GCNs variantes

➔ **GAT (Graph Attention Network)**

"Graph Attention Networks" by Velickovic et al. (2018).

Key Idea:

GAT introduces attention mechanisms to assign different weights to neighbors during aggregation, enabling nodes to selectively attend to more informative neighbors.



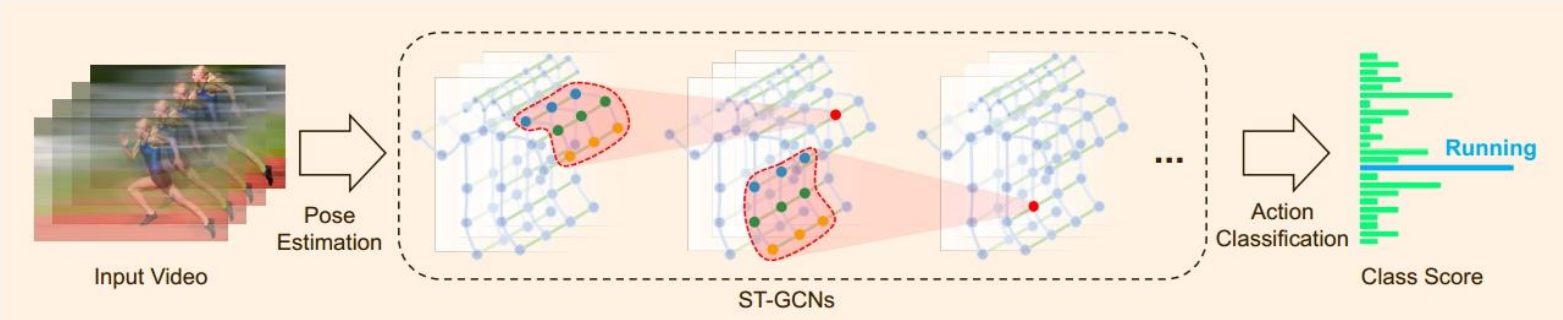
More advanced GCNs variantes

➔ **STGCN (Spatial Temporal Graph Convolutional Networks)**

"Spatial Temporal Graph Convolutional Networks for Skeleton-Based Action Recognition" by Yan et al. (2018).

Key idea:

STGCN incorporates both spatial and temporal dependencies within a graph structure. It applies graph convolutions across both spatial and temporal dimensions. This enables the model to capture spatial dependencies between different locations as well as temporal dependencies over time.



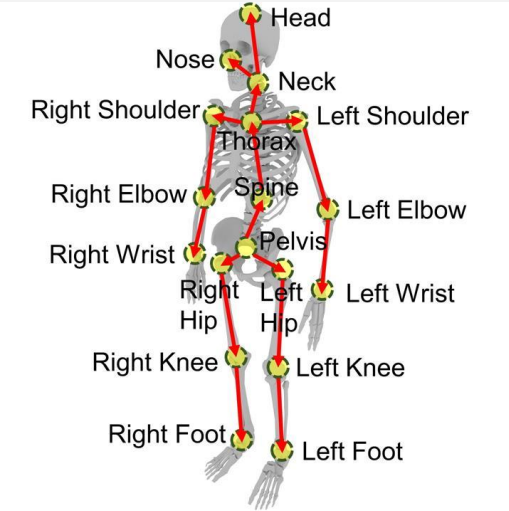
More advanced GCNs variantes

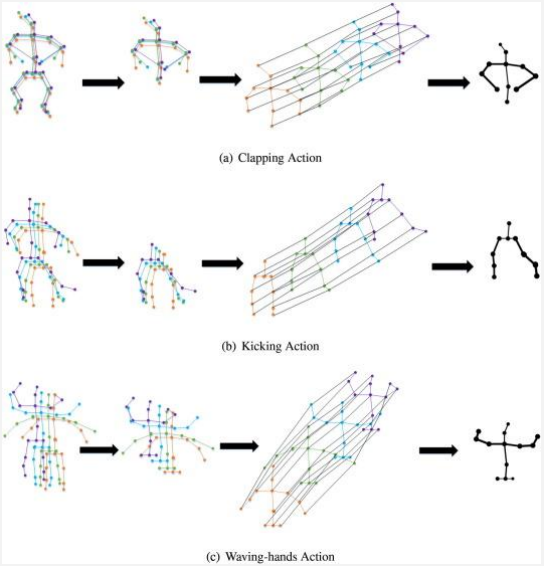
➔ **SGCN (Sparse Graph Convolution Network)**

"Sparse graph convolution network for pedestrian trajectory prediction" by Shi et al. (2021)

Key idea:

SGCNs are designed to be efficient for graphs with a sparse adjacency matrix. One of the main motivations for SGCNs is to reduce the computational complexity associated with standard GCNs, which involve matrix multiplications with the full adjacency matrix.





Challenges & Future Directions: What lies ahead

➔ Scalability

Challenge: As the size of the graph increases, the computational and memory requirements for training and inference increases.

Future Direction: Developing scalable GCN variants or exploring techniques to efficiently handle large-scale graphs.

➔ Handling Noisy or Incomplete Graphs

Challenge: Graphs in real-world applications often contain noise, missing edges, or incomplete information.

Future Direction: Developing GCN variants that are robust to noisy data or designing methods to impute missing information in graphs.

➔ Generalization to Heterogeneous Data

Challenge: Generalizing GCNs to handle heterogeneous graphs with different types of nodes and edges is a challenge.

Future Direction: the development of more flexible and adaptive architectures to handle heterogeneous graphs.

➔ Interpretable Graph Representations

Challenge: Understanding and interpreting the learned representations in GCNs can be challenging, especially in complex, high-dimensional graphs.

Future Direction: Exploring techniques for interpretability and visualization of graph representations.

Take home notes

- ➔ GCNs are used for learning on the graph.
- ➔ GCNs use both node features and the structure for the training
- ➔ The main idea of the GCN is to take the weighted average of all neighbors' node features (including itself): Lower-degree nodes get larger weights. Then, we pass the resulting feature vectors through a neural network for training.
- ➔ We can stack more layers to make GCNs deeper. Normally, we go for 2 or 3-layer GCN.

Additional Resources

- https://github.com/Jiakui/awesome-gcn
- https://antonsruberts.github.io/graph/gcn/
- https://ieeexplore.ieee.org/ielaam/5962385/9312808/9046288-aam.pdf
- https://distill.pub/2021/gnn-intro/
- https://distill.pub/2021/understanding-gnns/

Practical Example: GCN Training for Static Hand Gesture Recognition